(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 008 993 A2

(12) EUROPÄISCHE PATENTANMELDUNG

(43) Veröffentlichungstag:
14.06.2000 Patentblatt 2000/24

(51) Int. Cl.[7]: **G11C 29/00**

(21) Anmeldenummer: **99121604.5**

(22) Anmeldetag: **29.10.1999**

(84) Benannte Vertragsstaaten:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Benannte Erstreckungsstaaten:
AL LT LV MK RO SI

(30) Priorität: **30.10.1998 DE 19850115**

(71) Anmelder:
Infineon Technologies AG
81541 München (DE)

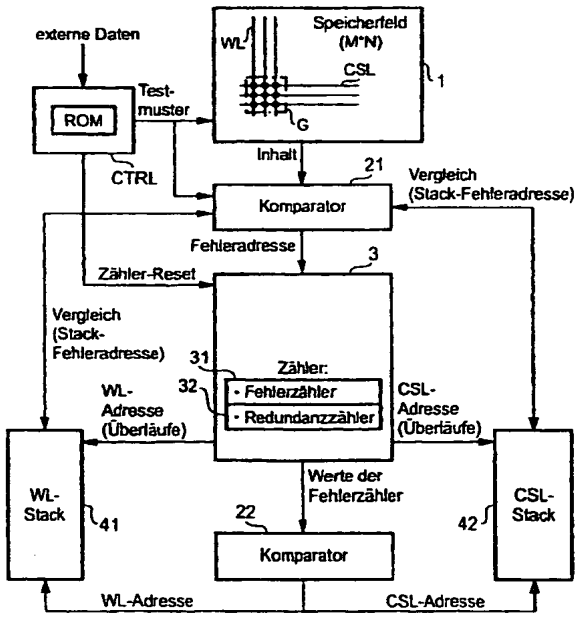
(72) Erfinder:
- **Lammers, Stefan**
  81739 München (DE)
- **Weber, Werner**
  80637 München (DE)

(74) Vertreter:
Zedlitz, Peter, Dipl.-Inf. et al
Patentanwalt,
Postfach 22 13 17
80503 München (DE)

(54) Schreib/Lesespeicher mit Selbsttestvorrichtung und zugehöriges Testverfahren

(57) Der Anmeldungsgegenstand betrifft einen Schreib-/Lesespeicher mit einer monolithisch integrierten Selbsttestvorrichtung, die ohne wesentliche externe Texthilfen iterativ einen Fehlertest mit einer Redundanzanalyse ermöglicht. Dies wird im wesentlichen dadurch erreicht, daß zu reparierende Wortleitungen gespeichert und von weiteren Untersuchungen ausgeschlossen werden und jeweils immer die Leitung mit den meisten bisher nicht erfaßten Fehlern ermittelt und zuerst untersucht werden, bis entweder die Anzahl der Reparaturleitungen nicht mehr ausreicht oder keine Fehler mehr auftreten.

FIG 1



EP 1 008 993 A2

Beschreibung

[0001] Die Erfindung betrifft Vorrichtungen und Verfahren zum Testen von Schreib-/Lesespeichern mit integrierter Redundanz, bei denen Testmuster in einem Speicherfeld eingeschrieben und anschließend ausgelesen und verglichen werden sowie, wenn möglich, solange Wortleitungen und Spaltenauswahlleitungen durch redundante Leitungen ersetzt werden, bis keine Fehler mehr vorhanden sind.

[0002] Bisher prüften auf dem Speicherchip integrierte Selbsttestarchitekturen nur, die Fehlerfreiheit des zugehörigen Chips. Eine Redundanzanalyse, bei der aus den Fehleradressen die zu ersetzenden Leitungen ermittelt werden, erfolgte hingegen bisher mit Hilfe eines externen Rechners, da bislang zunächst alle defekten Speicherzellen bestimmt und erst danach eine entsprechende Ersetzungsstrategie ermittelt wurde. Da ein derartiges Testverfahren unter anderem einen sehr großen Fehlerspeicher benötigt, ist dieses Verfahren nur mit einem externen Rechner sinnvoll durchführbar und kann praktisch nicht als eingebauter Selbsttest durchgeführt werden.

[0003] Die der Erfindung zugrundeliegende Aufgabe besteht nun darin, einen Schreib-/Lesespeicher mit einer integrierten Selbsttestvorrichtung und ein zugehöriges Testverfahren anzugeben, bei dem ein vollständiger Selbsttest inklusive der Redundanzanalyse ohne wesentliche externe Unterstützung durchführbar ist.

[0004] Die Erfindung wird nachfolgend anhand von in den Zeichnungen dargestellten Ausführungsbeispielen näher erläutert. Dabei zeigt

Figur 1 ein Blockdiagramm zur Erläuterung eines ersten Ausführungsbeispiels,

Figur 2 ein Blockdiagramm zur Erläuterung eines zweiten Ausführungsbeispiels und

Figur 3 ein Blockdiagramm zur Erläuterung eines dritten Ausführungsbeispiels.

[0005] Die Erfindung besteht im wesentlichen darin, daß ein Schreib-/Lesespeicher mit einer auf dem Chip befindlichen Selbsttestvorrichtung ohne wesentliche externe Testeinrichtungen dadurch getestet werden kann, daß nicht zunächst alle Fehler ermittelt werden und dann erst eine Redundanzanalyse erfolgt, sondern daß zu reparierende Leitungen gespeichert und von weiteren Untersuchungen ausgeschlossen werden und immer die Leitungen mit den meisten noch nicht erfaßten fehlerhaften Speicherzellen solange durch Redundanzleitungen virtuell ersetzt werden, bis entweder keine redundanten Leitungen oder keine fehlerhaften Zellen mehr vorhanden sind.

[0006] Figur 1 zeigt eine schematische Darstellung einer im Speicherchip integrierten Selbsttestarchitektur zur Reparatur eines Speicherbausteins unter Einsatz der vorhandenen Leitungsredundanz. Das Verfahren, das durch die Architektur realisiert wird ist iterativ, das heißt, es benötigt zur Ermittlung der zu aktivierenden Redundanzleitungen mehrere Testdurchläufe. Ein Testdurchlauf beinhaltet alle Testmuster, die für den entsprechenden Speicherchip während des sogenannten Prefuse-Tests vorgesehen sind. Von einer Steuereinheit CTRL die beispielsweise einen Nurlesespeicher ROM enthält, sind Testmuster in ein Speicherfeld einlesbar. Das Speicherfeld 1 entspricht hier einer Region eines Gesamtspeicherfeldes, dem eine gewisse Anzahl von redundanten Leitungen zugeordnet sind. Die Steuereinheit CTRL übernimmt die gesamte Ablaufsteuerung und kann mit externen Daten versorgt werden. Das Speicherfeld 1 weist eine Mehrzahl von Wortleitungen WL und eine Mehrzahl von Spaltenauswahlleitungen CSL auf und der Inhalt des Speicherfeldes wird durch einen Vergleicher 21 mit den eingeschriebenen Testmustern verglichen. Ferner führt der Vergleicher 21 einen Vergleich der Fehleradresse in Form von Wortleitung und Spaltenauswahlleitung mit in entsprechenden Stapelspeichern 41 und 42 gespeicherten Fehleradressen durch. Die im Vergleicher 21 festgestellte Fehleradresse steht Fehlerzählern 31 zur Verfügung. Darüber hinaus sind in der Selbsttestvorrichtung Redundanzzähler 32 vorgesehen. Werte der Fehlerzähler können in einem Vergleicher 22 verglichen werden.

[0007] In diesem Ausführungsbeispiel besitzt jede Wortleitung WL und jede Spaltenauswahlleitung einen eigenen Fehlerzähler. Zu Beginn des Testablaufs werden alle Fehlerzähler auf Null gesetzt. Während des ersten Testdurchlaufs werden die Wort- und Spaltenauswahlleitungen ermittelt, die eine besonders große Fehleranzahl aufweisen. Es wird für die Wortleitungen und die Spaltenauswahlleitungen eine Fehlerzahl bzw. eine Reparaturschwelle festgelegt, bei der die jeweilige Leitung durch eine entsprechend redundante Leitung ersetzt werden soll. Der Wert dieser Zahl ist vom zu testenden Speicherbaustein abhängig, da verschiedene Chips unterschiedliche Redundanzorganisationen besitzen können. Eine optimale Wahl dieser Reparaturschwelle kann für den jeweiligen zu testenden Chip in einfacher Weise experimentell ermittelt werden. Dabei muß auch der Flächenbedarf der Architektur berücksichtigt werden. Die Reparaturschwelle für die Wortleitungen kann von der der Spaltenauswahlleitungen verschieden sein.

[0008] Die Fehlerzähler werden z. B. vorteilhafterweise so dimensioniert, daß gerade im Falle des Überlaufs die Grenze erreicht wird, bei der eine Leitung durch eine entsprechende redundante Leitung ersetzt werden soll. Auf diese Weise kann sehr einfach das Überschreiten der verbliebenen Schwelle aufgrund eines Überlaufbits erfolgen.

[0009] Bevor die jeweiligen Fehlerzähler einer defekten Leitung erhöht werden, muß überprüft werden, ob die Wortleitungs- oder Spaltenauswahlleitungsadresse des Fehlers schon im zugehörigen Stapelspeicher abgelegt ist. In die-

sem Fall werden die Zähler nicht aktiviert, da der Fehler schon repariert wird. Dieser Vergleich muß schon im ersten Testdurchlauf erfolgen, weil ein Durchlauf aus mehreren Testmustern besteht. Dadurch wird vermieden, daß ein Fehler durch mehrere redundante Leitungen behoben wird. Die Adressen der Leitungen, deren Fehlerzähler nach dem ersten Testdurchlauf übergelaufen sind werden im Wortleitungsstapelspeicher 41 bzw. im Spaltenauswahlleitungsstapelspeicher 42 abgespeichert. Außerdem werden die zugehörigen Redundanzzähler 32 erhöht. Dazu muß die Domäne der zu ersetzenden Leitung bestimmt werden, denn es gibt üblicherweise für jede Wortleitungs- und für jede Spaltenauswahlleitungsdomäne eines Speicherchips eine fest vorgegebene Zahl an Redundanzleitungen. Das Speicherfeld 1 stellt eine Speicherregion dar, die wiederum aus WL- und CSL-Domänen besteht. Die Redundanzzähler sind vorteilhafterweise so ausgelegt, daß im Falle eines Überlaufs keine passende Redundanzleitung mehr vorhanden ist. Der Speicherbaustein ist also nicht reparierbar und der Test kann in diesem Fall sofort beendet werden.

[0010] Nachdem alle Testmuster einmal in den Schreib-/Lesespeicher eingeschrieben wurden, beginnt der zweite Teil des Verfahrens. Wiederholt werden die Testmuster an den Schreib-/Lesespeicher angelegt. Bei jedem Testdurchlauf werden alle Fehlerzähler auf Null gesetzt. Nun wird wiederum die Wortleitungs- und Spaltenauswahlleitungsadresse eines Fehlers mit den Einträgen der entsprechenden Stapelspeicher verglichen. Nur wenn keine der beiden Adressen in dem Stapelspeicher 41 und 42 abgelegt ist, werden die entsprechenden Fehlerzähler 31 erhöht. Nach einem Durchlauf wird mit Hilfe des Vergleichers 22 entschieden, welche Adresse, Wortleitung oder Spaltenauswahlleitung, im Stapelspeicher abgelegt wird. Dazu wird zunächst die Wortleitung und die Spaltenauswahlleitung mit der größten Fehleranzahl ermittelt. Dabei werden die Werte aller Fehlerzähler der Wortleitungen WL und die Fehlerzähler der Spaltenauswahlleitungen separat miteinander verglichen, wobei die Adresse der Wortleitung und der Spaltenauswahlleitung mit den meisten Fehlern ermittelt wird. Anschließend werden diese beiden Werte miteinander verglichen. Aus dem Vergleich geht hervor, welche der beiden Leitungen, also entweder die Wortleitung oder die Spaltenauswahlleitung, repariert werden soll. Existiert noch eine entsprechende Redundanzleitung in der betreffenden Domäne, ist also der Redundanzzähler dieser Domäne noch nicht übergelaufen, so wird die Adresse im Stapelspeicher abgelegt und der Redundanzzähler erhöht. Ansonsten wird die Adresse des anderen Maximums im Stapelspeicher gespeichert. Nur wenn auch für diese Leitung keine Redundanz mehr vorhanden ist, handelt es sich um einen irreparablen Speicherchip und der eingebaute Selbsttest kann abgebrochen werden. Beendet ist der Test, wenn der Speicherchip entweder keine Fehler mehr besitzt, daß heißt, daß alle Fehlerzähler nach einem Testdurchlauf auf Null sind, oder wenn keine passenden Redundanzleitungen vorhanden sind, also ein Redundanzzähler überläuft. Die Adressen der Leitungen, die ersetzt werden sollen, befinden sich nach einem vollständigen Testablauf in den Stapelspeichern 41 und 42. Dies sind sogenannte Fuse-Informationen, die der Laser benötigt, um den Chip mit der vorhandenen Redundanz reparieren zu können.

[0011] Um den Flächenbedarf dieser eingebauten Selbsttestarchitektur zu verringern, kann in einer vorteilhaften Ausgestaltung die Anzahl der Fehlerzähler reduziert werden. Das Verfahren wird hierbei nur unwesentlich verändert. Aus der Verringerung der Zähleranzahl resultiert eine höhere Testzeit, da eine größere Anzahl an Testdurchläufen notwendig ist. Denn wenn für jede Wortleitung und für jede Spaltenauswahlleitung ein Zähler vorhanden ist, kann mit dem oben beschriebenen Reparaturverfahren nach einem Testdurchlauf eine zu reparierende Leitung bestimmt werden. Stehen weniger Zähler zur Verfügung, so steigt die Anzahl der erforderlichen Testdurchläufe. Über einen Schalter kann den Fehlerzähler jeweils eine andere Gruppe G von Wortleitungen und Spaltenauswahlleitungen zugeordnet werden. Nach jedem Testdurchlauf wird dann die Wortleitung und die Spaltenauswahlleitungsadresse mit den bis dahin meisten Fehlern, sowie die zugehörigen Zählerwerte in einem weiteren Stapelspeicher abgespeichert. Dazu müssen alle neuen Zählerwerte mit dem bisherigen Maximalwert, welcher in einem Stapelspeicher abgelegt ist, verglichen werden. Dies erfolgt sowohl für die Wortleitungen als auch für die Spaltenauswahlleitungen. Danach werden die Zähler der nächsten Gruppe von Leitungen zugeordnet und es werden die Fehler auf diesen Leitungen gezählt.

[0012] In Figur 2 ist ein zweites Ausführungsbeispiel der Erfindung näher erläutert, wobei sich die Darstellungen im wesentlichen dadurch unterscheiden, daß zusätzlich ein Fehlerstapelspeicher 5 und ein Einzelbitstapelspeicher 43 vorhanden sind. Darüber hinaus ist angedeutet, daß der Testablauf unterbrochen werden muß und ein veränderter Testablauf erfolgen muß, wenn eine fehlerhafte Speicherzelle entdeckt wird.

[0013] Auch bei diesem Verfahren sind zur Auswahl der zu reparierenden Leitungen mehrere Testdurchläufe erforderlich. Dieses Verfahren unterscheidet sich von dem ersten Ausführungsbeispiel dadurch, daß der Testablauf bei diesem Konzept unterbrochen werden muß, wenn ein Fehler erkannt wird. Die Fehleradresse, also die entsprechende Wortleitungs- und Spaltenauswahlleitungsadresse, wird dann im Fehlerstapelspeicher abgespeichert. Der Vergleicher 22 überprüft anschließend, ob die Fehleradresse in einem der Stapelspeicher 41 ... 43 gespeichert ist, wobei im Speicher 43 Einzelbitfehler gespeichert werden. Ist dies der Fall, so wird der Testablauf mit einer anderen Zelle fortgesetzt. Ansonsten folgt ein veränderter Testablauf. Es wird hierbei separat die Anzahl aller Fehler auf der Wortleitung WL und auf der Spaltenauswahlleitung CSL des ursprünglichen Fehlers ermittelt. Vor dem Zählen der Fehler muß überprüft werden, ob die zusätzlichen Fehler nicht schon durch das Aktivieren einer redundanten Leitung behoben werden. Zuvor müssen die Fehlerzähler auf Null gesetzt werden. Da beim ersten Vergleich festgestellt wurde, daß sowohl die Adresse der Wortleitung als auch die Adresse der Spaltenauswahlleitung des ursprünglichen Fehlers noch nicht in

einem Stapelspeicher abgelegt ist, kann der folgende Vergleichsvorgang vereinfacht werden. Im Falle eines weiteren Fehlers auf der Wortleitung des ursprünglichen Fehlers muß lediglich der Stapelspeicher für die Spaltenauswahlleitungsadressen untersucht werden. Tritt ein zusätzlicher Fehler auf der Spaltenauswahlleitung des ursprünglichen Fehlers auf, so muß nur der Stapelspeicher die Wortleitungsadressen mit der Wortleitung des zusätzlichen Fehlers vergiichen werden. Übersteigt die Fehleranzahl eine bestimmte Grenze, so wird die Adresse der Leitung im entsprechenden Stapelspeicher abgelegt und der zugehörige Redundanzzähler erhöht. Die Grenze, bei der repariert werden soll, wird in jedem folgenden Testdurchlauf herabgesetzt.

[0014] Dies kann vorteilhafterweise dadurch geschehen, daß die unterschiedlichen Grenzen durch Zähler mit unterschiedlicher Wortbreite festgelegt werden. Beim Überlauf der jeweiligen Zähler wird diese Grenze erreicht. So können beispielsweise im ersten Testdurchlauf zwei Drei-Bit-Zähler verwendet werden. Dadurch würden dann in diesem Durchgang die Adressen der Leitungen abgespeichert, die mehr als sieben Fehler aufweisen. Im nächsten Testdurchlauf könnten zwei Zwei-Bit-Zähler zum Einsatz kommen, so daß die Grenze für eine erforderliche Reparatur von acht auf vier herabgesetzt wird. Im folgenden Testdurchgang können zwei Ein-Bit-Zähler benutzt werden. Die Aktivierung der verschiedenen Zähler erfolgt über Befehle, die beispielsweise im Speicher ROM der Steuereinheit gespeichert sind. Die Anzahl und die Dimensionierung der Zähler richtet sich nach dem zu testenden Speicherchip.

[0015] Durch die Anzahl der verwendeten Fehlerzähler wird auch die Zahl der erforderlichen Testdurchläufe vorgegeben. Gibt es keine weiteren Fehler auf der Wortleitung WL und der Spaltenauswahlleitung CSL des ursprünglichen Fehlers, so handelt es sich um einen Einzelbitfehler, dessen Adresse im Stapelspeicher 43 gespeichert wird. In ihm wird die komplette Adresse, also die Wortleitungs- und die Spaltenauswahlleitungsadresse abgespeichert. Am Ende des eingebauten Selbsttests wird dieser Stapelspeicher bearbeitet. Die Fehler, die dort gespeichert sind, können sowohl durch eine redundante Wortleitung als auch durch eine redundante Spaltenauswahlleitung ersetzt werden. Da die redundanten Wortleitungen meist anders organisiert sind als die redundanten Spaltenauswahlleitungen, unterscheiden sich die Testergebnisse, je nachdem mit welchen Redundanzleitungen die Einzelbitfehler behoben werden. Beendet ist der komplette Testablauf, wenn keine benötigten Redundanzleitungen mehr vorhanden sind, oder alle Fehler behoben sind. Läuft einer der Redundanzzähler über, so ist der Speicherchip nicht reparierbar und der Test kann sofort abgebrochen werden. Die vollständige Reparatur des Chips wird dadurch erkannt, daß während eines gesamten Testdurchgangs die Zähler immer auf Null stehenbleiben. Sobald ein Fehlerzähler erhöht wird, so besitzt der Chip noch mindestens einen Fehler und der Testablauf kann in diesem Fall nicht beendet werden.

[0016] In Figur 3 ist ein Blockdiagramm zur Erläuterung eines dritten Ausführungsbeispiels der Erfindung dargestellt, das sich im wesentlichen durch einen zusätzlichen Pufferspeicher 6 vom Blockdiagramm in Figur 2 unterscheidet. Der große Vorteil dieses Konzeptes besteht darin, daß nur ein einziger Testdurchlauf zur Bestimmung der zu ersetzenden Wortleitungen und Spaltenauswahlleitungen benötigt wird. Außerdem muß der Testablauf nicht abgeändert werden, was gewisse Nachteile mit sich bringen würde. Die Fehleradressen der jeweiligen Wortleitung werden in einem Pufferspeicher abgespeichert. Wird eine bestimmte Fehleranzahl auf dieser Wortleitung überschritten, so wird die Wortleitungsadresse im Stapelspeicher 41 abgelegt und der Inhalt des Pufferspeichers gelöscht. Außerdem muß der entsprechende Redundanzzähler 32 erhöht werden. Besitzt die Wortleitung weniger Fehler als die festgelegte Fehleranzahl, so wird der komplette Inhalt des Pufferspeichers, also alle gespeicherten Wortleitungs- und Spaltenauswahlleitungsadressen in den Fehlerstapelspeicher 5 eingeschrieben und anschließend der Pufferspeicher gelöscht. Die Größe des Fehlerstapelspeichers 5 beeinflußt die Testergebnisse. Während die nächste Wortleitung getestet wird, das heißt, die Fehleradressen bestimmt und in dem Pufferspeicher abgespeichert werden, erfolgt die Bearbeitung des Fehlerstapelspeichers. Es werden die Spaltenauswahlleitungen ermittelt und im Stapelspeicher 42 abgespeichert, die eine gewisse Fehleranzahl übertreffen. Dazu werden die entsprechenden CSL-Fehlerzähler 31 erhöht. Es gibt bei dieser Architektur für jede Spaltenauswahlleitung der zu testenden Speicherregion einen Fehlerzähler. Die Fehlerzähler für die jeweiligen Spaltenauswahlleitungen werden immer dann aktiviert, wenn neue Fehleradressen vom Pufferspeicher in den Fehlerspeicher eingeschrieben werden. Läuft einer dieser Fehlerzähler über, so wird die entsprechende Spaltenauswahlleitungsadresse im Stapelspeicher 42 abgespeichert. Außerdem müssen die zugehörigen Einträge, also die Spaltenauswahlleitungsadresse und die Wortleitungsadresse, aus dem Fehlerstapelspeicher gelöscht werden, da die Fehler nun durch den Einsatz einer redundanten Spaltenauswahlleitung repariert werden. Zudem wird der zugehörige Redundanzzähler erhöht. Diese Vorgehensweise sorgt dafür, daß auch bei Integration eines relativ kleinen Fehlerstapelspeichers schon akzeptable Testergebnisse erzielt werden. Läuft der Fehlerstapelspeicher über, so ist eine Reparatur des Chips mit Hilfe dieses Verfahrens nicht möglich und der eingebaute Selbsttest kann an dieser Stelle sofort abgebrochen werden. Nachdem der Testablauf beendet ist, muß der Inhalt des Fehlerstapelspeichers ausgewertet werden. Zur Auswertung des Fehlerstapelspeichers kann beispielsweise das im zweiten Ausführungsbeispiel erläuterte Verfahren verwendet werden.

Programmlistings als Anlage

Anhang A: C-Programme der entwickelten Konzepte

A.1 C-Programm des Konzeptes 1 (mit variabler Zähleranzahl)

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#define EIN_LAENGE 1000

int main (int argc,char *argv[])
{
const int N=2048;                          /* Row-Anzahl einer Region */
const int M=512;                           /* CSL-Anzahl einer Region */
const int NZCSL=1;                         /* Neu: Zaehleranzahl variierbar */
const int NZROW=4*NZCSL;                   /* Zaehleranzahl */
const int MAXERRCSL=8;                     /* Maximale CSL-Fehleranzahl */
const int MAXERRROW=8;                     /* Maximale WL-Fehleranzahl */
const int RROW=8;                          /* Anzahl redundanter Rows pro Domaene */
const int SCSL=8;                          /* Anzahl redundanter CSLs pro Domaene*/
const int FELDGROESSE=100000;              /* Anzahl Fehlereintraege pro Region */

int nirreg,
char *einpattern;
FILE *inpattern,*out;

void feldinit(long f[16] [FELDGROESSE])    /* Feldinitialisierung */
{
int i,j;
for (i=0;i<16;i++)
{
for (j=0;j<FELDGROESSE;j++)
{
f[i] [j]=0;
}
}
return;
}

void stackinit ( int stcsl[16] [17]        /* Stackinitialisierung */
                 ,int strow[16] [33])
{
int i,j;

for (i=0;i<16;i++)
{
for (j=0;j<17;j++)
{
stcsl[i] [j]=0;
}
}
for (i=0;i<16;i++)
{
for (j=0;j<33;j++)
{
strow[i] [j]=0;
}
}
return;
}
```


BNSDOCID: <EP_____1008993A2_I_>

```
void zaehlerinit ( int efcsl[NZCSL+1] [2]                    /* Zaehlerinitialisierung */
                   ,int efrow[NZROW+1] [2]
                   ,int hk)
{
int j;
for (j=1;j<NZROW+1;j++)
{
efrow[j] [0]=0;
efrow[j] [1]=4*hk+j-1;
}
for (j=1;j<NZCSL-1;j++)
{
efcsl[j] [0]=0;
efcsl[j] [1]=hk+j-1;
}
return;
}

void redundanzinit(int *zs                                   /* Redundanzinitialisierung */
                   ,int *zr)
{
zs[0]=0;
zs[1]=0;
zr[0]=0;
zr[1]=0;
zr[2]=0;
zr[3]=0;
return;
}

void zaehlerueberlauf(  long fehl                            /* Zaehlerueberlauf */
                        ,int efcsl[NZCSL+1] [2]
                        ,int efrow[NZROW-1] [2]
                        ,int *nc
                        ,int *nr
                        ,int region
                        ,int stcsl[16] [17]
                        ,int strow[16] [33]
                        ,int *zs
                        ,int *zr
                        ,int *al)
{
int rf,cf,j,i,l;
rf=(fehl/512);
cf=(fehl%512);
for (j=0;j<=(*nc);j++)                                       /* Vergleich mit Stack */
{
if(stcsl[region] [j-1]==cf)
return;
}
for (j=0;j<=(*nr);j++)
{
if[strow[region] [j+1]==rf)
return;
}
for (i=1;i<NZROW+1;i++)
{
if (efrow[i] [1]==rf)
{
if (efrow[i] [0]<MAXERRROW)
{
efrow[i] [0]=efrow[i] [0]+1;
goto M1;
}
else
{
if[zr[efrow[i] [1]/512]<RROW)
{
(*nr)++;
strow[region] [0]=*nr;
strow[region] [*nr]=efrow[i] [1];
zr[efrow[i] [1]/512]=zr[efrow[i] [1]/512]+1;
fprintf (out,"Ersatz von Row %4d in Domaene %2d mit mehr als %2d Fehlern\n",efrow[i] [1],efrow[i] [1]/512,efrow[i] [0]);
goto M1;
}
else
```

```
{
nirrreg++;
fprintf (out,"Region nicht reparierbar !\n\n");
*al=2;
return;
}
}
}
}
M1:
for (l=1;l<NZCSL+1;l++)
{
if (efcsl[l] [1]==cf)
{
if (efcsl[l] [0]<MAXERRCSL)
{
efcsl[l] [0]=efcsl[l] [0]+1;
return;
}
else
{
if(zs[efcsl[l] [1]/256]<SCSL)
{
(*nc)++;
stcsl[region] [0]=*nc;
stcsl[region] [*nc]=efcsl[l] [1];
zs[efcsl[l] [1]/256]=zs[efcsl[l] [1]/256]+1;
fprintf (out,"Ersatz von CSL %4d in Domaene %2d mit mehr als %2d Fehlern\n",efcsl[l] [1],efcsl[l] [1]/256,efcsl[l] [0]);
return;
}
else
{
nirrreg--;
fprintf (out,"Region nicht reparierbar !\n\n");
*al=2;
return;
}
}
}
}
return;
}

void fehlerzaehl( long fehl                                    /* Zaehler erhoehen */
                  ,int efcsl[NZCSL+1] [2]
                  ,int efrow[NZROW+1] [2]
                  ,int *nc
                  ,int *nr
                  ,int region
                  ,int stcsl[16] [17]
                  ,int strow[16] [33]
                  ,int *zs
                  ,int *zr
                  ,int *al)
{
int rf,cf,j,i,l,a,b;

a=0;
rf=(fehl/512);
cf=(fehl%512);
for (j=0;j<=(*nc);j++)
{
if(stcsl[region] [j+1]==cf)
return;
}
for (j=0;j<=(*nr);j++)
{
if(strow[region] [j+1]==rf)
return;
}
for (i=1;i<NZROW+1;i++)
{
if (efrow[i] [1]==rf)
{
efrow[i] [0]=efrow[i] [0]+1;
}
```

```
}
for (l=1;l<NZCSL-1;l++)
{
if (efcsl[l] [1]==ef)
{
efcsl[l] [0]=efcsl[l] [0]+1;
}
}
return;
}

void maxzachler( int efcsl[NZCSL+1] [2]                    /* Maximum im Zaehlerstack */
                 .int efrow[NZROW+1] [2])
{
int s,z,maxcsl,maxrow,snr,znr;

maxcsl=0;
maxrow=0;
snr=-1;
znr=-1;
for (s=0;s<NZCSL+1;s++)
{
if (efcsl[s] [0]>maxcsl)
{
maxcsl=efcsl[s] [0];
snr=efcsl[s] [1];
}
}
efcsl[0] [0]=maxcsl;
efcsl[0] [1]=snr;
for (z=0;z<NZROW+1;z++)
{
if (efrow[z] [0]>maxrow)
{
maxrow=efrow[z] [0];
znr=efrow[z] [1];
}
}
efrow[0] [0]=maxrow;
efrow[0] [1]=znr;
return;
}

void maxhilfstack(  int efcsl[NZCSL+1] [2]
                    .int efrow[NZROW+1] [2]
                    .int *al
                    .int *zs
                    .int *zr
                    .int stcsl[16] [17]
                    .int strow[16] [33]
                    .int *nc
                    .int *nr
                    .int region)
{
int j;

if ((efcsl[0] [0]==0)&&(efrow[0] [0]==0))
{
*al=0;
return;
}
else
{
if (efcsl[0] [0] != efrow[0] [0])
{
if (zs[efcsl[0] [1]/256]<SCSL)
{
zs[efcsl[0] [1]/256]=zs[efcsl[0] [1]/256]+1;
fprintf (out,"Ersatz von CSL %4d in Domaene %2d mit %2d Fehlern\n",efcsl[0] [1],efcsl[0] [1]/256,efcsl[0] [0]);
(*nc)++;
stcsl[region] [0]=*nc;
stcsl[region] [*nc]=efcsl[0] [1];
return;
}
else
{
```

```
if (zr[efrow[0] [1]/512]< RROW)
{
zr[efrow[0] [1]/512]=zr[efrow[0] [1]/512]+1;
fprintf (out,"Ersatz von Row %4d in Domaene %2d mit %2d Fehlern\n",efrow[0] [1],efrow[0] [1]/512,efrow[0] [0]);
(*nr)++;
strow[region] [0]=*nr;
strow[region] [*nr]=efrow[0] [1];
return;
}
else
{
nrrreg--;
fprintf (out,"Region nicht reparierbar !\n\n");
*al=0;
return;
}
}
}
else
{
if ((zr[efrow[0] [1]/512])< RROW)
{
zr[efrow[0] [1]/512]=zr[efrow[0] [1]/512]+1;
fprintf (out,"Ersatz von Row %4d in Domaene %2d mit %2d Fehlern\n",efrow[0] [1],efrow[0] [1]/512,efrow[0] [0]);
(*nr)++;
strow[region] [0]=*nr;
strow[region] [*nr]=efrow[0] [1];
return;
}
else
{
if (zs[efcsl[0] [1]/256]< SCSL)
{
zs[efcsl[0] [1]/256]=zs[efcsl[0] [1]/256]+1;
fprintf (out,"Ersatz von CSL %4d in Domaene %2d mit %2d Fehlern\n",efcsl[0] [1],efcsl[0] [1]/256,efcsl[0] [0]);
(*nc)++;
stcsl[region] [0]=*nc;
stcsl[region] [*nc]=efcsl[0] [1];
return;
}
else
{
nrrreg--;
fprintf (out,"Region nicht reparierbar !\n\n");
*al=0;
return;
}
}
}
}
return;
}

/********************** Hauptfunktion **********************/

void haupt(char *dateiein)
{
int errfcsl[NZCSL+1] [2];                    /* Zaehler und Hilfsstack (Erstes Element gibt */
int errfrow[NZROW+1] [2];                    /* Zaehlerhoehe an zweites Element Adresse) */
int stackcsl[16] [17];                       /* Spalten- bzw. */
int stackrow[16] [33];                       /* Zeilenstack */
int zrrow[4];                                /* Redundanzleitungszaehler (Row) */
int zscsl[2];                                /*          "                (CSL) */
int region;                                  /* Region in der Fehler auftritt */
int frow,fcsl;                               /* fehlerhafte Row bzw. CSL */
int nrrow,nscsl;                             /* Anzahl der Eintraege im Stack */
int n,i,j,k;
int algo;
int ndefreg;

long feld[16] [FELDGROESSE];                 /* Feld mit codierten Fehlern */
long fehler;                                 /* Codierte Fehleradresse */
char *ein,*regionptr,*outname,*einname;       /* Zeilenzeiger bzw. Hilfszeiger */
FILE *in;                                    /* Dateizeiger */

feldinit(feld);
```

```
stackinit(stackcsl,stackrow);
region=16;
fehler=0;
ndefreg=0;
outname=(char *)malloc(EIN_LAENGE);
einname=(char *)malloc(EIN_LAENGE);
strcpy(einname,"pattern/");
strncat(einname,dateiein,strlen(dateiein)-1);
strcpy(outname,"solalgo2/");
strncat(outname,dateiein,strlen(dateiein)-1);
strcat(outname,".solalgo");
printf("<%s> <%s>\n",einname,outname);
if (((in=fopen(einname,"r"))!=NULL)&&((out=fopen(outname,"w"))!=NULL))
{
ein=(char *)malloc(EIN_LAENGE);
regionptr=(char *)malloc(EIN_LAENGE);
while(fgets(ein,EIN_LAENGE,in)!=NULL)
{
regionptr=ein-1;
if (*regionptr=='S')
{
ndefreg--;
region=atoi(regionptr+6);
n=1;
}
else
{
if (region<16)
{
fehler=atoi(ein);
feld[region][n]=fehler;
feld[region][0]=n;
n--;
}
}
}
}
else
{
printf("Fehler beim Datei-Oeffnen !\n");
}
nnrreg=0;
for (i=0;i<16;i--)
{
if (feld[i][0]>0)
{
algo=1;
redundanzinit(zscsl,zrrow);
nrrow=0;
nscsl=0;
fprintf(out,"\n");
fprintf(out,"Auswertung Region %2d\n",i);
fprintf (out,"Zaehlerueberlaeufe :\n");
for (k=0;k<512;k=k+NZCSL)
{
zaehlerinit(errfcsl,errfrow,k);
for (j=1;j<=feld[i][0];j++)
{
zaehlerueberlauf(feld[i][j],errfcsl,errfrow,&nscsl,&nrrow,i,stackcsl,stackrow,zscsl,zrrow,&algo);
if (algo==2)
goto M2;
}
}
M2:
fprintf (out,"Die uebrigen Fehler :\n");
while (algo--1)
{
errfcsl[0][0]=0;
errfcsl[0][1]=0;
errfrow[0][0]=0;
errfrow[0][1]=0;
for (k=0;k<512;k=k+NZCSL)
{
zaehlerinit(errfcsl,errfrow,k);
for (j=1;j<=feld[i][0];j++)
{
```


EPNSDOCID: <EP_____1008993A2_I_>

```
fehlerzaehl(feld[i] [j].errfcsl.errfrow,&nscsl,&nrrow,i,stackcsl,stackrow,zscsl,zrrow,&algo);
}
maxzaehler(errfcsl,errfrow);
}
maxhilfstack(errfcsl,errfrow,&algo,zscsl,zrrow,stackcsl,stackrow,&nscsl,&nrrow,i);
}
}
}
fprintf(out,"\n\n");
fprintf(out,"%2d defekte Regionen\n",ndefreg);
if (nirrreg>0)
fprintf(out,"%2d Regionen nicht reparierbar\n\n",nirrreg);
fclose(in);
fclose(out);
}                                                                    /* main function */

/************************** Hauptprogramm ********************************/

if ((inpattern=fopen(argv[1],"r"))!=NULL)
{
einpattern=(char *)malloc(EIN_LAENGE);
while(fgets(einpattern,EIN_LAENGE,inpattern)!=NULL)
{
hauptteinpattern);
}
}
else
printf("Fehler beim Datei-Oeffnen !\n");
}                                                                    /* main */
```

A.2 C-Programm des Konzeptes II

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#define EIN_LAENGE 1000

int main (int argc,char *argv[])
{
const int N=2048;                                   /* Row-Anzahl einer Region */
const int M=512;                                    /* CSL-Anzahl einer Region */
const int MAXERRCSL=8;                              /* Maximale CSL-Fehleranzahl */
const int MAXERRROW=8;                              /* Maximale WL-Fehleranzahl */
const int RROW=8;                                   /* Anzahl redundanter Rows pro Domaene */
const int SCSL=8;                                   /* Anzahl redundanter CSLs pro Domaene */
```

```
const int FELDGROESSE=100000;                    /* Anzahl Fehlereintraege pro Region */

int nimreg,ndefreg;
char *einpattern;
FILE *inpattern,*out;

void feldinit(long f[16] [FELDGROESSE])          /* Feldinitialisierung */
{
int i,j;

for (i=0;i<16;i--)
{
for (j=0;j<FELDGROESSE;j++)
{
f[i] [j]=0;
}
}
return;
}

void stackinit ( int stcsl[16] [17]              /* Stackinitialisierung */
                ,int strow[16] [33])
{
int i,j;

for (i=0;i<16;i--)
{
for (j=0;j<17;j--)
{
stcsl[i] [j]=0;
}
}
for (i=0;i<16;i--)
{
for (j=0;j<33;j--)
{
strow[i] [j]=0;
}
}
return;
}

void redundanzinit(int zs[2]                     /* Redundanzinitialisierung */
                  ,int zr[4])
{
zs[0]=0;
zs[1]=0;
zr[0]=0;
zr[1]=0;
zr[2]=0;
zr[3]=0;
return;
}

void einzelstackinit(long eist[49])              /* Einzelstack Initialisierung */
{
int j;

for (j=0;j<49;j--)
{
eist[j]=0;
}
return;
}

void stackvergleich( long fehl                   /* Fehleradresse mit Stack vergleichen */
                    ,int stcsl[16] [17]
                    ,int strow[16] [33]
                    ,int region
                    ,int *v
                    ,long eist[49])
{
int j,rf,cf;

(*v)=0;
```

```
rf=(fehl/512);
cf=(fehl%512);
for (j=1;j<=eist[0];j++)
{
if (eist[j]==fehl)
{
(*v)=1;
return;
}
else
(*v)=0;
}
for (j=1;j<=strow[region] [0];j++)
{
if (rf==strow[region] [j])
{
(*v)=1;
return;
}
else
(*v)=0;
}
for (j=1;j<=stcsl[region] [0];j++)
{
if (cf==stcsl[region] [j])
{
(*v)=1;
return;
}
else
(*v)=0;
}
return;
}

void vergleich( long fehl                                   /* Zusaetzliche Fehler suchen */
                long f[16] [FELDGROESSE]
                int stcsl[16] [17]
                int strow[16] [33]
                int region
                int *zc
                int *zr)
{
int j,k,rf,cf;

(*zc)=0;
(*zr)=0;
rf=(fehl/512);
cf=(fehl%512);
for (j=1;j<=f[region] [0];j++)
{
if (rf==((f[region] [j])/512))
{
for (k=1;k<=stcsl[region] [0];k++)
{
if (stcsl[region] [k]==(f[region] [j])%512)
goto M1;
}
(*zr)++;
}
M1:
if (cf==((f[region] [j])%512))
{
for (k=1;k<=strow[region] [0];k++)
{
if (strow[region] [k]==(f[region] [j])/512)
goto M2;
}
(*zc)++;
}
M2:
}
return;
}

void einstack(  long eist[49]                               /* Einzelbitfehler ersetzen */
```

```
                    .int zs[2]
                    .int zr[4]
                    .int region
                    .int *nc
                    .int *nr
                    .int *al
                    .int stcsl[16] [17]
                    .int strow[16] [33])
{
int j,rf,cf;

for (j=1;j<=eist[0];j++)
{
rf=eist[j]/512;
cf=eist[j]%512;
if (zs[cf/256]<SCSL)
{
zs[cf/256]++;
fprintf (out,"Ersatz von CSL %4d in Domaene %2d \n",cf,cf/256);
(*nc)++;
stcsl[region] [0]=*nc;
stcsl[region] [*nc]=cf;
}
else
{
if (zr[rf/512]< RROW)
{
zr[rf/512]++;
fprintf (out,"Ersatz von Row %4d in Domaene %2d \n",rf,rf/512);
(*nr)++;
strow[region] [0]=*nr;
strow[region] [*nr]=rf;
}
else
{
nrrreg++;
fprintf (out,"Region nicht reparierbar !\n\n");
*al=0;
return;
}
}
}
return;
}

/********************** Hauptfunktion ********************************/

void haupt(char *dateiein)
{
int zfcsl;                                  /* CSL-Fehler*/
int zfrow;                                  /* Row-Fehler */
int stackcsl[16] [17];                      /* Spalten- bzw. */
int stackrow[16] [33];                      /* Zeilenstack */
int zrrow[4];                               /* Redundanzleitungszaehler (Row) */
int zscsl[2];                               /* Redundanzleitungszaehler (CSL) */
int region;                                 /* Region in der Fehler auftritt */
int nrrow,nscsl,neinst;                     /* Anzahl der Eintraege im Stack */
int n,i,j,k;
int algo;
int MAXCSL,MAXROW;
int frow,fcsl;
int ver;

long einzelstack[49];                       /* Einzelbitstack */
long feld[16] [FELDGROESSE];                /* Feld mit codierten Fehlern */
long fehler;                                /* Codierte Fehleradresse */
char *ein,*regionptr,*outname,*einname;     /* Zeilenzeiger bzw. Hilfszeiger */
FILE *in;                                   /* Dateizeiger */

feldinit(feld);
stackinit(stackcsl,stackrow);
region=16;
fehler=0;
ndefreg=0;
outname=(char *)malloc(EIN_LAENGE);
einname=(char *)malloc(EIN_LAENGE);
```

```
strcpy(einname,"pattern/");
strncat(einname,dateiein,strlen(dateiein)-1);
strcpy(outname,"solalgo3/");
strncat(outname,dateiein,strlen(dateiein)-1);
strcat(outname,".solalgo");
printf("<%s> <%s>\n",einname,outname);
if (((in=fopen(einname,"r"))!=NULL)&&((out=fopen(outname,"w"))!=NULL))
{
ein=(char *)malloc(EIN_LAENGE);
regionptr=(char *)malloc(EIN_LAENGE);
while(fgets(ein,EIN_LAENGE,in)!=NULL)
{
regionptr=ein+1;
if (*regionptr=='S')
{
ndefreg++;
region=atoi(regionptr+6);
n=1;
}
else
{
if (region<16)
{
fehler=atoi(ein);
feld[region][n]=fehler;
feld[region][0]=n;
n++;
}
}
}
}
else
{
printf("Fehler beim Datei-Oeffnen !\n");
}

********** Beginn des Algorithmus **********

nurreg=0;
stackinit(stackcsl,stackrow);
for (i=0;i<16;i++)
{
if (feld[i][0]>0)
{
algo=1;
neinst=0;
nrrow=0;
nscsl=0;
MAXCSL=MAXERRCSL;
MAXROW=MAXERRROW;
redundanzinit(zscsl,zrrow);
einzelstackinit(einzelstack);
fprintf(out,"\n\n");
fprintf(out,"Auswertung Region %2d\n",i);
fprintf(out,"\n");
fprintf(out,"Mehrfache Fehler: \n");
M1:
while ((algo==1)&&((MAXROW>0)&&(MAXCSL>0)))
{
for (j=1;j<=feld[i][0];j++)
{
stackvergleich(feld[i][j],stackcsl,stackrow,i,&ver,einzelstack);
if (ver==0)
{
frow=feld[i][j]/512;
fcsl=feld[i][j]%512;
vergleich(feld[i][j],feld,stackcsl,stackrow,i,&zfcsl,&zfrow);
if ((zfcsl==1)&&(zfrow==1))
{
neinst++;
einzelstack[neinst]=feld[i][j];
einzelstack[0]=neinst;
}
else
{
if (zfrow>=MAXROW)
```

```
{
if(zrrow[frow/512]<RROW)
{
nrrow++;
stackrow[i] [0]=nrrow;
stackrow[i] [nrrow]=frow;
(zrrow[frow/512])++;
fprintf (out,"Ersatz von Row %4d in Domaene %2d mit %4d Fehlern\n",frow,frow/512,zfrow);
zfcsl--;
}
else
{
nirrreg++;
algo=0;
fprintf (out,"Region nicht reparierbar !\n\n");
goto M1;
}
}
if (zfcsl>=MAXCSL)
{
if(zscsl[fcsl/256]<SCSL)
{
nscsl++;
stackcsl[i] [0]=nscsl;
stackcsl[i] [nscsl]=fcsl;
(zscsl[fcsl/256])++;
fprintf (out,"Ersatz von CSL %4d in Domaene %2d mit %4d Fehlern\n",fcsl,fcsl/256,zfcsl);
}
else
{
nirrreg++;
algo=0;
fprintf (out,"Region nicht reparierbar !\n\n");
goto M1;
}
}
}
}
}
MAXROW=MAXROW-1;                                    /* Neue Grenze fuer Fehleranzahl */
MAXCSL=MAXCSL-1;
}
fprintf (out,"Einzelbitfehler: \n");
einstack(einzelstack,zscsl,zrrow,i,&nscsl,&nrrow,&algo,stackcsl,stackrow);
}
}
fprintf(out,"\n\n");
fprintf(out,"%2d defekte Regionen\n",ndefreg);
if (nirrreg>0)
fprintf(out,"%2d Regionen nicht reparierbar\n\n",nirrreg);
fclose(in);
fclose(out);
}                                                   /* main function */

/************************** Hauptprogramm **************************/

if ((inpattern=fopen(argv[1],"r"))!=NULL)
{
einpattern=(char *)malloc(EIN_LAENGE);
while(fgets(einpattern,EIN_LAENGE,inpattern)!=NULL)
{
haupt(einpattern);
}
}
else
printf("Fehler beim Datei-Oeffnen !\n");
}                                                   /* main */
```

A.3 C-Programm des Konzeptes III

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define EIN_LAENGE 1000

int main (int argc,char *argv[])
{
const int N=2048;                          /* Row-Anzahl einer Region */
const int M=512;                           /* CSL-Anzahl einer Region */
const int MAXERRCSL=8;                     /* Maximale CSL-Fehleranzahl) */
const int MAXERRROW=8;                     /* Maximale WL-Fehleranzahl */
const int RROW=8;                          /* Anzahl redundanter Rows pro Domaene */
const int SCSL=8;                          /* Anzahl redundanter CSLs pro Domaene */
const int FSTACKGROESSE=25;                /* Anzahl Eintraege im Fehlerstack */
const int FELDGROESSE=100000;              /* Anzahl Fehlereintraege pro Region */

int nimreg;
char *einpattern;
FILE *inpattern,*out;

void feldinit(long f[16][FELDGROESSE])     /* Feldinitialisierung */
{
int i,j;

for (i=0;i<16;i++)
{
for (j=0;j<FELDGROESSE;j++)
f[i][j]=0;
}
return;
}
```

```
void bufferinit(long b[9])                          /* Buffer initialisieren */
{
int i;
b[0]=0;
for (i=1;i<9;i++)
b[i]=-1;
return;
}

void fstackinit (long fst[FSTACKGROESSE+1])         /* Fehlerstack initialisieren */
{
int i;

fst[0]=0;
for (i=1;i<=FSTACKGROESSE;i++)
fst[i]=-1;
return;
}

void stackinit ( int stcsl[16] [17]                 /* Stackinitialisierung */
                 ,int strow[16] [33])
{
int i,j;

for (i=0;i<16;i++)
{
for (j=0;j<17;j++)
{
stcsl[i] [j]=0;
}
}
for (i=0;i<16;i++)
{
for (j=0;j<33;j++)
{
strow[i] [j]=0;
}
}
return;
}

void zaehlerinit (int *efcsl)                       /* Zaehlerinitialisierung */
{
int j;

for (j=0;j<M;j++)
{
efcsl[j]=0;
}
return;
}

void redundanzinit(int *zs                          /* Redundanzinitialisierung */
                   ,int *zr)
{
zs[0]=0;
zs[1]=0;
zr[0]=0;
zr[1]=0;
zr[2]=0;
zr[3]=0;
return;
}

void einzelstackinit(long eist[49])                 /* Einzelstack initialisieren */
{
int j;

for (j=0;j<49;j++)
{
eist[j]=0;
}
return;
}
```

```
void finbuffer (long f[16] [FELDGROESSE]                    /* Fehler im Buffer ablegen */
                ,long b[9]
                ,int zr[4]
                ,int strow[16] [33]
                ,int region
                ,int row
                ,int *nb
                ,int *al
                ,int *nr)
{
int i,j;

*nb=0;
for (i=1;i<=f[region] [0];i++)
{
if (f[region] [i]/512==row)
{
if ((*nb)< MAXERRROW)
{
(*nb)++;
b[0]=*nb;
b[*nb]=f[region] [i];
}
else
{
if(zr[row/512] >= RROW)
{
(*nr)++;
strow[region] [0]=*nr;
strow[region] [*nr]=row;
zr[row/512]=zr[row/512]-1;
fprintf (out,"Ersatz von Row %4d in Domaene %2d mit mehr als %2d Fehlern\n",row ,row/512 ,MAXERRROW);
b[0]=0;
*nb=0;
b[0]=0;
for (j=1;j<=9;j++)
b[j]=-1;
return;
}
else
{
nirreg++;
fprintf (out,"Region nicht reparierbar !\n\n");
(*al)=0;
return;
}
}
}
}
return;
}

void fehlerstack( long b[8]                                 /* Buffer in Fehlerstack */
                  ,int *nb
                  ,long fstack[FSTACKGROESSE+1]
                  ,int *nfst
                  ,int zcsl[512]
                  ,int *zs
                  ,int stcsl[16] [17]
                  ,int *al
                  ,int *nc
                  ,int region)
{
int i,j,k,nh;
long hstack[FSTACKGROESSE+1];

for (i=1;i<=b[0];i++)
{
if (fstack[0]< FSTACKGROESSE)
{
(*nfst)++;
fstack[0]=*nfst;
fstack[*nfst]=b[i];
for (j=1;j<=stcsl[region] [0];j++)
{
if (stcsl[region] [j]==b[i]*512)
```

```
{
fstack[*nfst]=-1;
(*nfst)--;
fstack[0]=(*nfst);
goto M1;
}
}
if (zcsl[b[i]%512]<MAXERRCSL)
(zcsl[b[i]%512])++;
else
{
if(zs[(b[i]%512)/256]<SCSL)
{
(*nc)++;
stcsl[region] [0]=*nc;
stcsl[region] [*nc]=b[i]%512;
(zs[(b[i]%512)/256])++;
fprintf (out,"Ersatz von CSL %4d in Domaene %2d mit mehr als %2d Fehlern \n",(b[i]%512), (b[i]%512)/256,MAXERRCSL);
for (j=1;j<=fstack[0];j++)
{
if (fstack[j]%512==b[i]%512)
fstack[j]=-1;
}
}
else
{
nutreg--;
(*al)=0;
fprintf (out,"Region nicht reparierbar !\n\n");
return;
}
}
}
else
{
nutreg--;
fprintf (out,"FSUE: Region nicht reparierbar !\n");
(*al)=0;
return;
}
M1:
}
nh=0;
hstack[0]=0;
for (j=1;j<=FSTACKGROESSE;j++)
hstack[j]=-1;
for (k=1;k<=fstack[0];k++)
{
if (fstack[k]!=-1)
{
nh++;
hstack[nh]=fstack[k];
hstack[0]=nh;
}
}
fstack=hstack;
(*nfst)=nh;
return;
}

void stackvergleich( long fehl                    /* Fehleradresse mit Stack vergleichen */
                     ,int stcsl[16] [17]
                     ,int strow[16] [33]
                     ,int region
                     ,int *v
                     ,long eist[49])
{
int j,rf,cf;

(*v)=0;
rf=(fehl/512);
cf=(fehl%512);
for (j=1;j<=eist[0];j++)
{
if (eist[j]==fehl)
{
```

```
(*v)=1;
return;
}
else
(*v)=0;
}
for (j=1;j<=strow[region] [0];j++)
{
if (rf==strow[region] [j])
{
(*v)=1;
return;
}
else
(*v)=0;
}
for (j=1;j<=stcsl[region] [0];j++)
{
if (cf==stcsl[region] [j])
{
(*v)=1;
return;
}
else
(*v)=0;
}
return;
}

void vergleich( long fehl                                   /* Zusaetzliche Fehler suchen */
                long f[FSTACKGROESSE+1]
                int stcsl[16] [17]
                int strow[16] [33]
                int region
                int *zc
                int *zr)
{
int j,k,rf,cf;

(*zc)=0;
(*zr)=0;
rf=(fehl/512);
cf=(fehl%512);
for (j=1;j<=f[0];j++)
{
if (rf==((f[j])/512))
{
for (k=1;k<=stcsl[region] [0];k++)
{
if (stcsl[region] [k]==(f[j])%512)
goto M1;
}
(*zr)++;
}
M1:
if (cf==((f[j])%512))
{
for (k=1;k<=strow[region] [0];k++)
{
if (strow[region] [k]==(f[j])/512)
goto M2;
}
(*zc)++;
}
M2:
}
return;
}

void einstack(  long eist[49]                               /* Einzelbitfehler ersetzen */
                int zs[2]
```

```
                    .int zr[4]
                    .int region
                    .int *nc
                    .int *nr
                    .int *al
                    .int stcsl[16] [17]
                    .int strow[16] [33])
{
int j,rf,cf;

for (j=1;j<=eist[0];j++)
{
rf=eist[j]/512;
cf=eist[j]%512;
if (zr[rf/512]<RROW)
{
zr[rf/512]++;
fprintf (out,"Ersatz von Row %e4d in Domaene %e2d \n",rf,rf/512);
(*nr)++;
strow[region] [0]=*nr;
strow[region] [*nr]=rf;
}
else
{
if (zs[cf/256]<SCSI.)
{
zs[cf/256]++;
fprintf (out,"Ersatz von CSL %e4d in Domaene %e2d \n",cf,cf/256);
(*nc)++;
stcsl[region] [0]=*nc;
stcsl[region] [*nc]=cf;
}
else
{
nirreg--;
fprintf (out,"Region nicht reparierbar !\n\n");
*al=0;
return;
}
}
}
return;
}

/*************************** Hauptfunktion ****************************/

void haupt(char *dateiein)
{
int errfcsl[M];                         /* CSL-Fehlerfeld */
int zfcsl,zfrow;                        /* Fehlerzaehler Row bzw. CSL */
int stackcsl[16] [17];                  /* Spalten bzw. */
int stackrow[16] [33];                  /* Zeilenstack */
int zrrow[4];                           /* Redundanzleitungszaehler (Row) */
int zscsl[2];                           /* Redundanzleitungszaehler (CSL) */
int region;                             /* Region in der Fehler auftritt */
int frow,fcsl;                          /* fehlerhafte Row bzw. CSL */
int nrrow,nscsl,nbuffer,nfstack_neinst; /* Anzahl der Eintraege im Stack bzw. Buffer */
int MAXCSL,MAXROW;
int n,i,j,k,l;
int algo;
int ndefreg;
int ver;

long fstack[FSTACKGROESSE+1];           /* Row-Fehlerfeld */
long einzelstack[49];                   /* Einzelbitstack */
long feld[16] [FELDGROESSE];            /* Feld mit codierten Fehlern */
long buffer[8];                         /* Buffer fuer 7 Fehleradressen */
long fehler;                            /* Codierte Fehleradresse */
char *ein,*regionptr,*outname,*einname; /* Zeilenzeiger bzw. Hilfszeiger */
FILE *in;                               /* Dateizeiger */

feldinit(feld);
stackinit(stackcsl,stackrow);
region=16;
fehler=0;
ndefreg=0;
```

```
outname=(char *)malloc(EIN_LAENGE);
einname=(char *)malloc(EIN_LAENGE);
strcpy(einname,"pattern");
strncat(einname,dateiein,strlen(dateiein)-1);
strcpy(outname,"solalgo5/");
strncat(outname,dateiein,strlen(dateiein)-1);
strcat(outname,"_solalgo");
printf("<%s><%s>\n",einname,outname);
if (((in=fopen(einname,"r"))!=NULL)&&((out=fopen(outname,"w"))!=NULL))
{
ein=(char *)malloc(EIN_LAENGE);
regionptr=(char *)malloc(EIN_LAENGE);
while(fgets(ein,EIN_LAENGE,in)!=NULL)
{
regionptr=ein+1;
if (*regionptr=='S')
{
ndefreg++;
region=atoi(regionptr+6);
n=1;
}
else
{
if (region<16)
{
fehler=atoi(ein);
feld[region][n]=fehler;
feld[region][0]=n;
n++;
}
}
}
}
else
{
printf("Fehler beim Datei-Oeffnen !\n");
}
numreg=0;
for (i=0;i<16;i++)
{
if (feld[i][0]>0)
{
fstackinit (fstack);
redundanzinit(zscsl,zzrow);
zaehlerinit(errfcsl);
einzelstackinit(einzelstack);
neinst=0;
MAXCSL=4;
MAXROW=4;
nrrow=0;
nscsl=0;
nbuffer=0;
nfstack=0;
algo=1;
fprintf(out,"\n");
fprintf(out,"Auswertung Region %2d\n",i);
fprintf(out,"Mehrfache Fehler: \n");
for (j=0;j<2048;j++)
{
bufferinit(buffer);
finbuffer(feld,buffer,zzrow,stackrow,i,j,&nbuffer,&algo,&nrrow);
if (algo==0)
goto M1;
if (buffer[0]>0)
{
fehlerstack (buffer,&nbuffer,fstack,&nfstack,errfcsl,zscsl,stackcsl,&algo,&nscsl,i);
if (algo==0)
goto M1;
}
}
while ((algo==1)&&((MAXROW>0)&&(MAXCSL>0)))
{
for (j=1;j<=fstack[0];j++)
{
stackvergleich(fstack[j],stackcsl,stackrow,i,&ver,einzelstack);
if (ver==0)
```

```
{
frow=fstack[j]/512;
fcsl=fstack[j]%512;
vergleich(fstack[j],fstack,stackcsl,stackrow,i,&zfcsl,&zfrow);
if ((zfcsl==1)&&(zfrow==1))
{
neinst--;
einzelstack[neinst]=fstack[j];
einzelstack[0]=neinst;
}
else
{
if (zfrow !=MAXROW)
{
if(zrrow[frow/512]!=RROW)
{
nrrow--;
stackrow[i] [0]=nrrow;
stackrow[i] [nrrow]=frow;
(zrrow[frow/512])--;
fprintf (out,"Ersatz von Row %e4d in Domaene %e2d mit %e4d Fehlern\n",frow,frow/512,zfrow);
zfcsl--;
}
else
{
nirrreg++;
algo=0;
fprintf (out,"Region nicht reparierbar !\n");
goto M1;
}
}
if (zfcsl !=MAXCSL)
{
if(zscsl[fcsl/256]!=SCSL)
{
nscsl--;
stackcsl[i] [0]=nscsl;
stackcsl[i] [nscsl]=fcsl;
(zscsl[fcsl/256])--;
fprintf (out,"Ersatz von CSL %e4d in Domaene %e2d mit %e4d Fehlern\n",fcsl,fcsl/256,zfcsl);
}
else
{
nirrreg--;
algo=0;
fprintf (out,"Region nicht reparierbar !\n");
goto M1;
}
}
}
}
}
MAXROW=MAXROW/2;                                        /* Neue Grenze fuer Fehleranzahl */
MAXCSL=MAXCSL/2;
}
fprintf (out,"Einzelbitfehler: \n");
einstack(einzelstack,zscsl,zrrow,i,&nscsl,&nrrow,&algo,stackcsl,stackrow);
}
M1:
}
fprintf(out,"\n\n");
fprintf(out,"%e2d defekte Regionen\n",ndefreg);
if (nirrreg !=0)
fprintf(out,"%e2d Regionen nicht reparierbar\n",nirrreg);
fclose(in);
fclose(out);
return;
}                                                       /* main function */

/****************************** Hauptprogramm ******************************/

if ((inpattern=fopen(argv[1],"r"))!=NULL)
{
einpattern=(char *)malloc(EIN_LAENGE);
while(fgets(einpattern,EIN_LAENGE,inpattern)!=NULL)
{
```

```
haupt(einpattern);
}
}
else
printf("Fehler beim Datei-Oeffnen !\n");
}                                                    /* main */
```

Patentansprüche

1. Schreib-/Lesespeicher mit mindestens einem Speicherfeld (1) und einer Selbsttestvorrichtung,

   bei dem die Selbsttestvorrichtung zusammen mit dem mindestens einem Speicherfeld monolithisch integriert ist und Fehlerzähler (31) für Wortleitungen (WL) und Spaltenauswahlleitungen (CSL), Redundanzverbrauchszähler (32), Stapelspeicher (41, 42) für zu reparierende Wortleitungen und Spaltenauswahlleitungen, Vergleicher (21, 22) und eine Steuereinheit (CTRL) aufweist und
   bei dem Vergleicher mit den Stapelspeichern für zu reparierende Wortleitungen und Spaltenauswahlleitungen und Vergleicher mit Fehlerzählern für Wortleitungen (WL) und Spaltenauswahlleitungen (CSL) verbunden sind.

2. Schreib-/Lesespeicher nach Anspruch 1,

   bei dem die Steuereinheit (CTRL) derart ausgebildet ist, daß der Reihe nach

   a) zunächst alle Fehlerzähler und Redundanzverbrauchszähler auf Null gesetzt werden,
   b) mit Hilfe der Vergleicher ein Vergleich zwischen einer in das Speicherfeld eingeschriebenen und aus diesem ausgelesenen Information erfolgt und eine fehlerhafte Zelle festgestellt wird,
   c) die zur fehlerhaften Zelle gehörige Wortleitung und Spaltenauswahlleitung überprüft wird, ob sie im Stapelspeicher bereits gespeichert ist und nur dann der zur jeweiligen Wortleitung und jeweiligen Spaltenauswahlleitung gehörige Zähler erhöht wird, wenn die Wortleitung oder die Spaltenauswahlleitung noch nicht gespeichert wurde,
   d) die Wortleitung und/oder die Spaltenauswahlleitung im Stapelspeicher abgespeichert und der zugehörige Redundanzverbrauchszähler erhöht werden, wenn die zugehörigen Fehlerzähler eine Reparaturschwelle überschreiten,
   e) die Schritte b) bis d) für alle vorgegebenen Testmuster und alle Zellen des Speicherfeldes erfolgen oder bis einer der Redundanzverbrauchszähler überläuft,
   f) alle Fehlerzähler auf Null gesetzt werden,
   g) der Schritt b) durchgeführt wird,
   h) der Schritt c) erfolgt,
   i) mit Hilfe der Vergleicher und der Fehlerzähler getrennt die Wortleitung und die Spaltenauswahlleitung mit der größten Fehlerzahl ermittelt wird,
   j) entweder die Wortleitung oder die Spaltenauswahlleitung mit der größten Fehlerzahl ausgewählt wird,
   k) diese im Stapelspeicher gespeichert und der zugehörige Redundanzverbrauchszähler inkrementiert wird, falls der zugehörige Redundanzverbrauchszähler noch nicht übergelaufen ist,
   l) mit der anderen der nach Schritt i) ausgewählten Leitung Schritt k) durchgeführt wird, wenn für die nach Schritt j) ausgewählte Leitung der zugehörige Redundanzverbrauchszähler übergelaufen ist,
   m) der Test abgebrochen wird, wenn der Redundanzverbrauchszähler überläuft,
   n) die Schritte f) bis m) für alle vorgegebenen Testmuster und für alle Speicherzellen des Speicherfeldes wiederholt werden und
   o) Schritt n) solange wiederholt wird, bis alle Fehlerzähler gleich Null bleiben oder einer der Redundanzverbrauchszähler überläuft.

3. Schreib-/Lesespeicher nach Anspruch 2,

bei dem weniger Fehlerzähler als Wortleitungen und Spaltenauswahlleitungen und ein zusätzlicher Speicher für ein maximum und eine zugehörige Fehlerzahl vorhanden sind und die Fehlerzähler über Schalter verschiedenen Gruppen (G) von Wortleitungen und Spaltenauswahlleitungen zuordenbar sind und bei dem die Steuereinheit derart vorhanden ist, daß mit Hilfe der Vergleicher die Wortleitungen und Spaltenauswahlleitungen mit der größten Fehlerzahl innerhalb einer Gruppe ermittelt und im Stapelspeicher anstelle der Wortleitungen und Spaltenauswahlleitungen mit der bisher größten Fehlerzahl gespeichert werden, wenn die Fehlerzahl größer ist als die bisherige größte Fehlerzahl und daß dies anstelle aller Zellen nur die zur jeweiligen Gruppe von Wortleitungen und Spaltenauswahlleitungen gehörigen Zellen getestet werden und der Test für alle Gruppen des Speicherzellenfeldes wiederholt wird.

4. Schreib-/Lesespeicher nach Anspruch 2 oder 3,

bei der in Schritt j) diejenige der beiden Leitungen ausgewählt wird, die die größte Fehlerzahl aufweist.

5. Schreib-/Lesespeicher nach Anspruch 1,

bei dem die Fehlerzähler (31) mit unterschiedlicher Wortbreite nur für eine Wortleitung (WL) und eine Spaltenauswahlleitung (CSL), zusätzlich ein Stapelspeicher (43) für Einzelbitfehler und ein Fehlerstapelspeicher (5) vorhanden sind und die Steuereinheit (CTRL) derart ausgebildet ist, daß der Reihe nach

a) zunächst alle Fehlerzähler und Redundanzverbrauchszähler gleich Null gesetzt werden,
b) mit Hilfe der Vergleicher ein Vergleich zwischen den in das Speicherfeld eingeschriebenen Informationen und ausgelesenen Informationen erfolgt und die fehlerhafte Zelle festgestellt und im Fehlerstapelspeicher gespeichert werden,
c) die zur fehlerhaften Zelle gehörige Wortleitung (WL) als auch die Spaltenauswahlleitung (CSL) mit Hilfe eines der Vergleicher überprüft wird, ob sie im Stapelspeicher, als fehlerhafte Wortleitung, fehlerhafte Spaltenauswahlleitung oder als Einzelbitfehler bereits gespeichert ist und die Schritte d) und e) übergangen werden, wenn die Wortleitung und/oder die Spaltenauswahlleitung oder der Einzelfehler bereits gespeichert sind,
d) zur fehlerhaften Speicherzelle (Z) gehörige im Fehlerstapelspeicher gespeicherte Wortleitungen und Spaltenauswahlleitungen separat auf weitere Fehler überprüft werden und der zur Wortleitung gehörige Fehlerzähler bzw. der zur Spaltenauswahlleitung gehörige Fehlerzähler nur dann erhöht werden, wenn der jeweilige weitere Fehler noch nicht in einem Stapelspeicher gespeichert ist,
e) Wortleitungen und/oder Spaltenauswahlleitungen im entsprechenden Stapelspeicher gespeichert sowie der zughörige Redundanzverbrauchszähler erhöht werden, wenn der zugehörige Fehlerzähler eine Reparaturschwelle überschreitet,
f) alle Fehlerzähler gleich Null gesetzt werden und die Schritte b) bis e) für alle Testmuster und alle Speicherzellen des Speicherfeldes wiederholen und
g) die Reparaturschwelle senken durch einen Fehlerzähler mit geringerer Wortbreite sowie die Schritt f) solange wiederholen, bis bei der niedrigsten Reparaturschwelle die Fehlerzähler für alle Testmuster und alle Speicherzellen gleich Null bleiben oder der Redundanzzähler überläuft.

6. Schreib-/Lesespeicher nach Anspruch 5,

bei dem im Schritt d) weitere Fehler in der Wortleitung (WL) nur im Stapelspeicher (42) für die zu reparierenden Auswahlleitungen und weitere Fehler in der zugehörigen Spaltenauswahlleitung (CSL) nur im Stapelspeicher (41) für die Wortleitungen gesucht werden.

7. Schreib-/Lesespeicher nach Anspruch 2,

bei dem zusätzlich ein Pufferspeicher (6) vorhanden ist und eine Steuereinheit (CTRL) derart vorhanden ist, daß zunächst, vor den Schritten a) bis g), der Reihe nach

A) die fehlerhaften Zellen einer Wortleitung (WL) mit Wortleitung (WL) und Spaltenauswahlleitung (CSL) im Pufferspeicher gespeichert werden,
B) geprüft wird, ob die Fehleranzahl innerhalb einer Wortleitung eine durch die Größe des Pufferspeichers vorgegebene Reparaturschwelle überschreitet und, wenn die Reparaturschwelle nicht überschritten wird, der Pufferinhalt in den Fehlerstapelspeicher übernommen und der Puffer gelöscht wird,

C) eine fehlerhafte Wortleitung in den Wortleitungsstapelspeicher übernommen wird, der Pufferspeicher gelöscht und der entsprechende Redundanzverbrauchszähler erhöht wird, wenn die Reparaturschwelle überschritten wurde, und

D) während die nächste Wortleitung durch die Schritte A) bis C) getestet wird, die im Fehlerstapelspeicher befindlichen Fehler der vorhergehenden Wortleitung bearbeitet, wobei der für die jeweilige Spaltenauswahlleitung vorgesehene Fehlerzähler (31) erhöht wird und im Falle eines Zählerüberlaufs betreffende Spaltenauswahlleitung im Stapelspeicher (42) für fehlerhafte Spaltenauswahlleitungen gespeichert und der entsprechende Redundanzverbrauchszähler (32) erhöht wird,

E) die Schritte A) bis D) für alle vorgegebenen Testmuster und alle Zellen des Speicherfeldes wiederholen oder abbrechen und bereits als nicht reparierbar melden, wenn der Fehlerstapelspeicher oder einer der Redundanzverbrauchszähler überläuft.

8. Verfahren zum Testen eines Schreib-/Lesespeichers mit einer Selbsttestvorrichtung,

bei dem zu reparierende Wortleitungen(WL) und/oder Spaltenauswahlleitungen (CSL) gespeichert und deren Zellen für weitere Untersuchungen ausgeschlossen werden,
bei dem die Wortleitungen und / oder die Spaltenauswahlleitungen mit den meisten noch nicht bereits von den gespeicherten Wortleitungen und / oder Spaltenauswahlleitungen erfaßten fehlerhaften Speicherzellen ermittelt und vor den anderen Wortleitungen und / oder die Spaltenauswahlleitungen untersucht werden und
bei dem festgestellt wird, ob die Anzahl der redundanten Leitungen noch ausreicht und ob bereits keine fehlerhaften Zellen mehr vorhanden sind.

9. Verfahren zum Testen eines Schreib-/Lesespeichers nach Anspruch 8, bei dem

a) zunächst Fehlerzähler und Redundanzverbrauchszähler auf Null gesetzt werden,
b) ein Vergleich zwischen einer in das Speicherfeld eingeschriebenen und aus diesem ausgelesenen Information erfolgt und eine fehlerhafte Zelle festgestellt wird,
c) die zur fehlerhaften Zelle gehörige Wortleitung und Spaltenauswahlleitung überprüft wird, ob sie bereits gespeichert ist und nur dann der zur jeweiligen Wortleitung oder jeweiligen Spaltenauswahlleitung gehörige Fehlerzähler erhöht wird, wenn die Wortleitung oder die Spaltenauswahlleitung noch nicht gespeichert wurde,
d) die Wortleitung und/oder die Spaltenauswahlleitung abgespeichert und der zugehörige Redundanzverbrauchszähler erhöht werden, wenn die zugehörigen Fehlerzähler eine Reparaturschwelle überschreiten,
e) die Schritte b) bis d) für alle vorgegebenen Testmuster und alle Zellen des Speicherfeldes erfolgen oder bis die Redundanz verbraucht ist,
f) alle Fehlerzähler auf Null gesetzt werden,
g) der Schritt b) durchgeführt wird,
h) der Schritt c) erfolgt,
i) mit Hilfe der Vergleicher und der Fehlerzähler getrennt die Wortleitung und die Spaltenauswahlleitung mit der größten Fehlerzahl ermittelt wird,
j) entweder die Wortleitung oder die Spaltenauswahlleitung mit der größten Fehlerzahl ausgewählt wird,
k) diese gespeichert und der Redundanzverbrauchszähler inkrementiert wird, falls noch benötigte Redundanz vorhanden ist,
l) mit der anderen der nach Schritt i) ausgewählten Leitung Schritt k) durchgeführt wird, wenn für die nach Schritt j) ausgewählte Leitung keine Redundanz mehr vorhanden ist,
m) der Test abgebrochen wird, wenn die entsprechende Redundanz verbraucht ist,
n) die Schritte f) bis m) für alle vorgegebenen Testmuster und für alle Speicherzellen des Speicherfeldes wiederholt werden und
o) Schritt n) solange wiederholt wird, bis keine Fehler mehr vorhanden sind oder die Redundanz aufgeraucht ist.

10. Verfahren zum Testen eines Schreib-/Lesespeichers nach Anspruch 8, bei dem

a) zunächst alle Fehlerzähler und alle Redundanzverbrauchszähler gleich Null gesetzt werden,
b) ein Vergleich zwischen den in das Speicherfeld eingeschriebenen Informationen und ausgelesenen Informationen erfolgt und die fehlerhafte Zelle festgestellt und gespeichert werden,
c) die zur fehlerhaften Zelle gehörige Wortleitung (WL) als auch die Spaltenauswahlleitung (CSL) überprüft wird, ob sie als fehlerhafte Wortleitung, fehlerhafte Spaltenauswahlleitung oder als Einzelbitfehler bereits gespeichert ist und die Schritte d) und e) übergangen werden, wenn sie bereits gespeichert ist,

d) zur fehlerhaften Speicherzelle (Z) gehörige zwischengespeicherte Wortleitungen und Spaltenauswahlleitungen separat auf Fehler überprüft werden und der zur Wortleitung gehörige Fehlerzähler bzw. der zur Spaltenauswahlleitung gehörige Fehlerzähler nur dann erhöht werden, wenn der jeweilige Fehler noch nicht gespeichert ist,

e) Wortleitungen und/oder Spaltenauswahlleitungen gespeichert sowie der entsprechendes Redundanzverbrauchszähler erhöht werden, wenn der zugehörige Fehlerzähler eine Reparaturschwelle überschreitet,

f) zunächst alle Fehlerzähler gleich Null gesetzt und die Schritte b) bis e) für alle Testmuster und alle Speicherzellen des Speicherfeldes wiederholen werden und

g) die Reparaturschwelle gesenkt sowie die Schritt f) solange wiederholt werden, bis, bei der niedrigsten Reparaturschwelle, für alle Testmuster und alle Speicherzellen keine Fehler mehr auftreten oder einer der Redundanzverbrauchszähler überläuft.

11. Verfahren zum Testen eines Schreib-/Lesespeichers nach Anspruch 10, bei dem daß zunächst, vor den Schritten a) bis g), der Reihe nach

A) die fehlerhaften Zellen einer Wortleitung (WL) mit Wortleitung (WL) und Spaltenauswahlleitung (CSL) in einem Püfferspeicher gespeichert werden,

B) geprüft wird, ob die Fehleranzahl innerhalb einer Wortleitung eine Reparaturschwelle überschreitet und, wenn die Reparaturschwelle nicht überschritten wird, der Inhalt des Pufferspeichers in einen Fehlerstapelspeicher übernommen und der Pufferspeicher gelöscht wird,

C) eine fehlerhafte Wortleitung gespeichert wird, der Puffer gelöscht und der entsprechende Redundanzverbrauchszähler erhöht wird, wenn die Reparaturschwelle überschritten wurde, und

D) während die nächste Wortleitung durch die Schritte A) bis C) getestet wird, die im Fehlerstapelspeicher befindlichen Fehler der vorhergehenden Wortleitung bearbeitet, wobei der für die jeweilige Spaltenauswahlleitung vorgesehene Fehlerzähler (31) erhöht wird und beim Überschreiten einer Reparaturschwelle betreffende Spaltenauswahlleitung gespeichert und der entsprechende Redundanzverbrauchszähler (32) erhöht wird,

E) die Schritte A) bis D) für alle vorgegebenen Testmuster und alle Zellen des Speicherfeldes wiederholen oder abbrechen und bereits als nicht reparierbar melden, wenn der Fehlerstapelspeicher oder einer der Redundanzzähler überläuft.

FIG 1

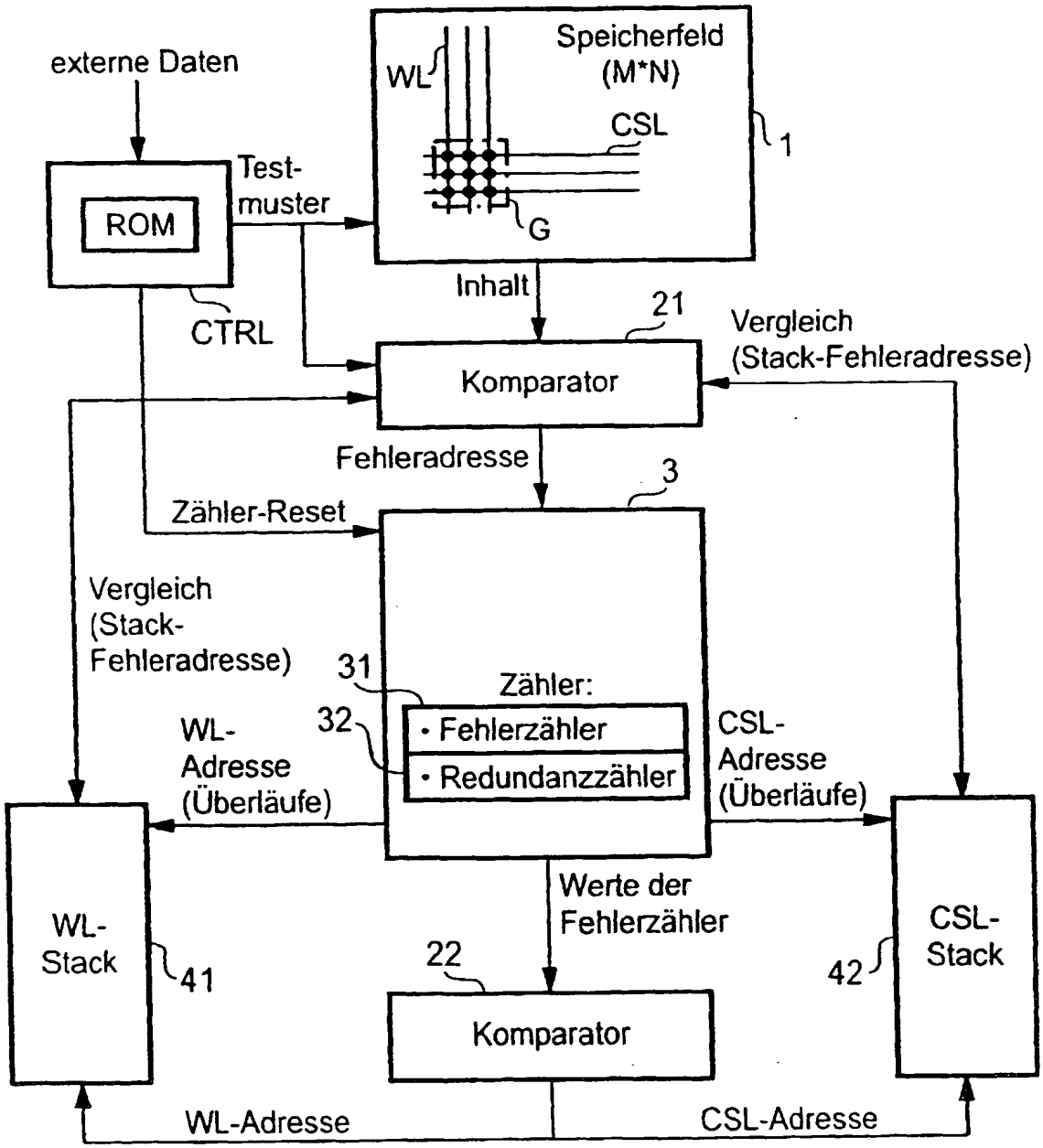


FIG 2

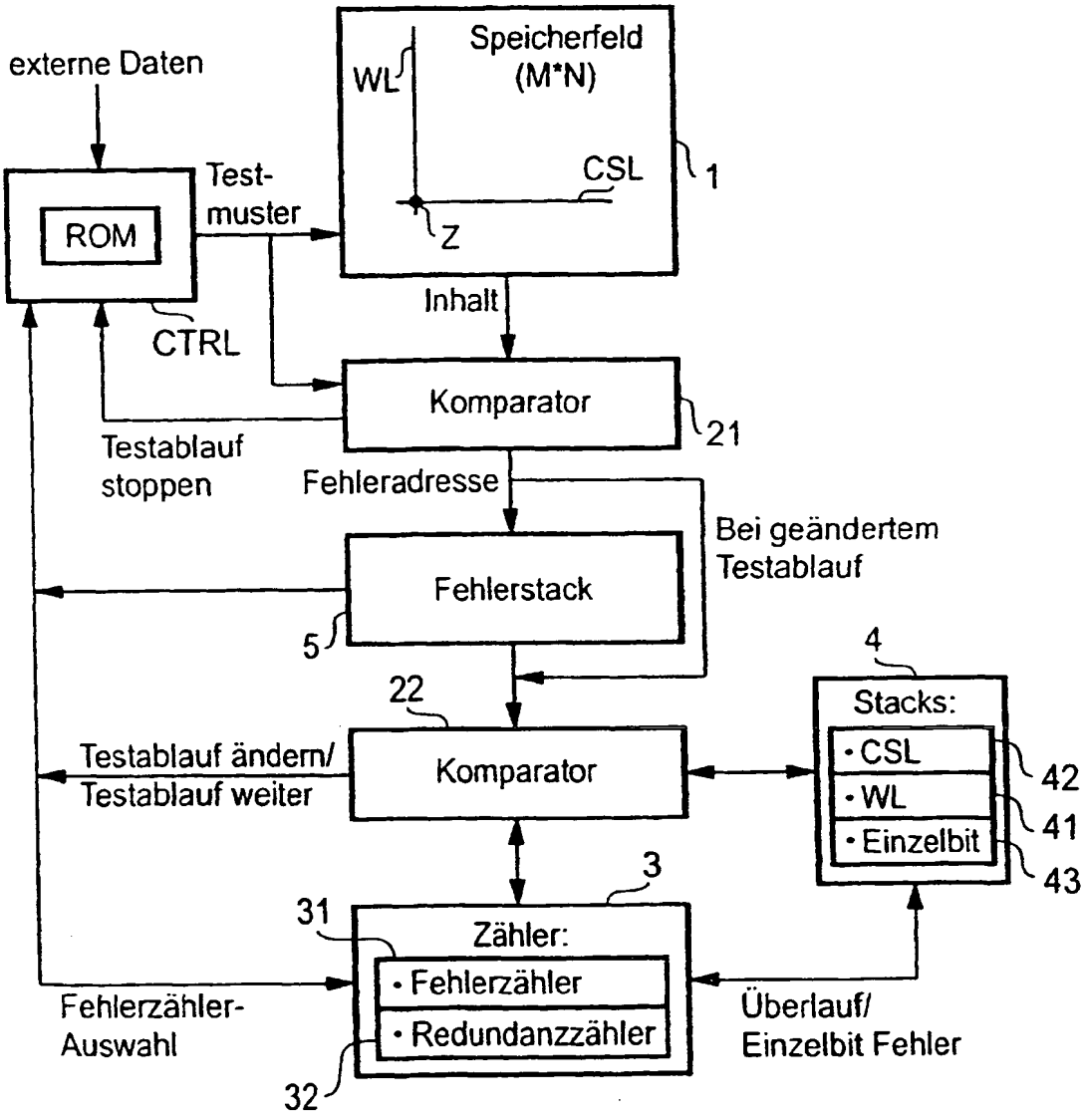
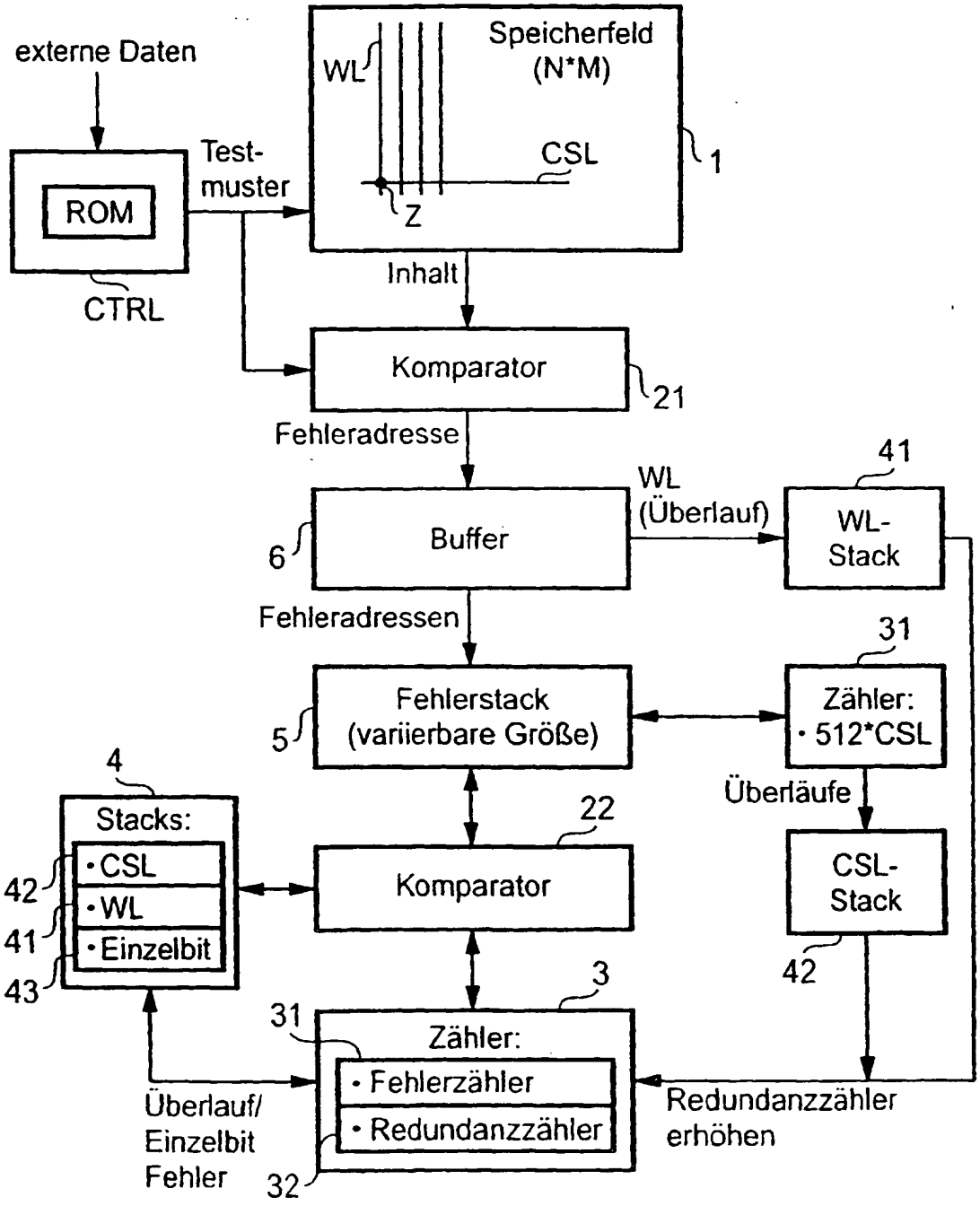


FIG 3



This Page Blank (upto)